# The ARM11 Architecture

Ian Davey

Payton Oliveri

Spring 2009

CS433

## Why ARM Matters

- Over 90% of the embedded market is based on the ARM architecture
- ARM Ltd. makes over $100 million USD annually in royalties and licensing fees for this technology
- Over two billion units are shipped each year
- We will focus primarily on the ARM1176JZF-S, which is used in a number of smartphones as well as the iPod Touch

## General Overview

- ARM stands for Advanced RISC Machine
- The ARM11 is based on the ARMv6 instruction set architecture
- Bi-endian – can operate in either little-endian or big-endian format
  - Most devices today use little-endian
- Actually uses two instruction sets – the 32-bit ARM and the 16-bit Thumb

## ARM and Thumb

- Since many embedded devices have small amounts of memory, a smaller, 16-bit instruction set can be used
- This 16-bit 'Thumb” instruction set makes use of implied operands and reduced functionality to reduce code size
- Thumb instructions are decoded into ARM instructions on the fly at execution time, though consuming one additional cycle
- CPU is either in ”ARM state” or ”Thumb state”

## Registers – 32-bit ARM mode

- 16 general-purpose registers R0-R15
  - R13 is the stack pointer and is often called SP
  - R14 holds return addresses and is often called LR (for link register)
  - R15 is the program counter and is often called PC
  - PC is always word-aligned
- 17 general-purpose ”mode-specific” registers (used for exception handling, etc.)
- 7 status registers, one for each operating mode

# Registers – 16-bit Thumb mode

- 7 sets of 11 registers each
  - 8 general-purpose registers R0-R7
  - Stack pointer, link register, and program counter
- Each set is for a different operating mode
- More on operating modes later

## Status Register Specifics

- Bits 0 through 4 determine the processor operating mode
- Bit 5 indicates whether the processor is in ARM or Thumb state
- Bits 6 and 7 disable interrupts
- Bits 28 through 31 are ALU condition code flags
  - N for negative ALU result
  - Z for zero ALU result
  - C for overflow after shift operation
  - V for overflow after signed arithmetic operation

## Operating Modes

- As mentioned before, each mode has its own mode-specific registers, including a status register
- The 8 modes of operation consist of 7 ”priviledged modes” - which are used to handle exceptions and ease normal resource restrictions – and 1 ”user mode” which is used during normal operation and has all restrictions in place

## Operating Modes

1. User – normal operation
2. Fast interrupt – handling of ”fast” interrupts
3. Interrupt – handling of all other interrupts
4. Supervisor – operating system protected mode
5. Abort – abortion of memory access
6. System – operating system privileged mode
7. Undefined – invalid instruction in stream
8. Secure monitor – on-chip security features

## ARM Instruction Set Architecture

- Each instruction is 32 bits long
- Highest four bits determine condition (indicated in status register) under which the instruction is executed
  - Can discard instruction immediately after decode
  - Only two pipeline stages are wasted (as seen next)
  - Fewer branch instructions needed, smaller code
- Other fields contain operands, offset constants, and various 1-bit flags

## The Pipeline

- 8 stages in normal pipeline
  1. Fe1 – Address is sent and instruction received
  2. Fe2 – Much of the branch prediction goes here
  3. De – Decode instruction
  4. Iss – Read registers and issue instruction
  5. Sh – Perform shift operations
  6. ALU – Perform integer operations
  7. Sat – Saturate results
  8. WB – Write back data to registers

## The Shift Pipeline Stage

- There are no explicit shift instructions
- Each arithmetic instruction has a field to specify amount to shift one operand
- ARM microarchitecture contains a barrel shifter that can perform shifts and rotates on operands
- This is why the status register has separate flags of shift and arithmetic overflow

## Alternate Pipeline Paths

- Sh, ALU, and Sat can be replaced with three MAC stages, which perform multiplication operations
- These three stages can also be replaced with different stages for memory operations:
  1. ADD – calculate address
  2. DC1 and DC2 – Access data cache

## Parallelism within Instructions

- Some instructions, such as those which access memory and increment a register at the same time (useful for array operations), will use both the memory access pathway and the arithmetic pathway simultaneously
- If the data cache misses and there is a stall in the memory access pathway, the arithmetic pathway will continue execution anyway
- This frees up the ALU and saturation stage for use with other instructions

## Branches

- Branch instruction contains condition field, link flag, and 24-bit offset field
- Target must be word-aligned, so offset is effectively 26 bits
- Target address calculated by sign-extending PC-8+offset (subtract 8 for pipeline)
- Program can branch to addresses up to 32 megabytes away from PC
- If the link flag is set, store old PC in LR

## Branch Prediction

- While the condition attached to every instruction helps reduce costly explicit branches, it does not eliminate them entirely
- Naive approach in ARM9 is to always predict that the branch will not be taken
  - This risks losing 3+ cycles if wrong!
  - This was more acceptable for the ARM9, which has a shorter pipeline
- Therefore, ARM11 implements more aggressive techniques

## Branch Target Address Cache

- Direct-mapped, 128-entry cache which keeps track of previous branch instructions (indexed by address) and their results
- Stores 2-bit prediction history, which it uses to make next prediction
- Lines only evicted from cache in the event of a conflict with another branch address
- Very effective for loops, where branches have the same result many times in a row

## Static Branch Prediction

- Used when BTAC entry not available
- Configured in hardware to take branches with negative offsets, and not take branches with positive offsets
- Once again this would help loop performance, since jumps backwards tend to follow the body of a loop
- The forward-not-taken choice accommodates the way compilers handle conditionals

## Branch Folding

- Removes predicted branch instructions from the instruction stream
- Branches with ”side effects” are not subject to this optimization, since those effects (such as links into new procedures) must be carried out
- Branches to branches cannot be folded, for reasons that will become clear shortly

# What happens on misprediction?

- Pipeline is flushed and correct instructions are fetched
- Instructions following folded branches fail
- Whenever there is a chance a folded branch has been mispredicted, address of the alternative choice must be remembered
- These alternatives are retrieved if it turns out the branch was indeed mispredicted
- This is why branches to branches cannot be folded – would need to store multiple alternatives

## Memory Access

- Data must be moved to registers before it can be manipulated
- A memory word can be indexed by a register plus or minus a 12-bit offset constant
- A halfword or byte can be indexed the same way, except the offset can only be 8 bits
- Support for block-data transfer – can transfer up to 16 registers to and from memory in a single instruction

## Memory Hierarchy

- L1 cache involves separate instruction and data caches and a write buffer
  - Each cache is 4-way set-associative, ranging from 4KB to 64KB in size, with 8-word cache lines
  - Cache is virtually indexed, virtually tagged
  - Data cache misses are non-blocking
  - Upon eviction, if data needs to be written back to memory, the line is added to the write buffer
  - Write buffer handles all RAW hazards that may occur when holding needed data

## Memory Hierarchy

- L2 cache is off-chip
  - Instruction controller, data controller, and DMA hardware on-chip each has its own 64-bit wide port, allowing for simultaneous accesses to the cache
  - An additional 32-bit peripheral interface connects to processor peripherals such as coprocessors

## Translation Lookaside Buffer

- Actually has two levels
- First level is known as Micro-TLB
  - 2 10-entry, fully associative TLBs, one for I-cache, one for D-cache
  - Performs translation in parallel
- Second level is called the Main TLB
  - Next level up when Micro-TLB misses
  - 64-entry, 2-way set associative buffer
  - Also has 8-entry, fully associative section, used mainly in Secure Monitor mode

## Hardware Stack

- ARM also supports a hardware stack
- Both upward and downward-growing stacks are supported
  - Which direction to grow is specified by the flags in the push and pop instructions
- Stack is used to store activation records

## ARM Calling Conventions

- R0 through R3 hold the first four arguments
  - Additional arguments are passed in reverse order on the stack
  - Values larger than 32 bits are passed as multiple 32-bit values
  - Floating-point values are passed in registers on the floating-point coprocessor
- R4 through R10 hold local variables
- R11 serves as the frame pointer

## ARM Calling Conventions

- R0 through R3 hold return values
  - Once again, data types larger than 32 bits are treated as multiple 32-bit values
- R12 holds intra-procedural intermediate values as well as serves as a scratchpad register between calls

## Exception Handling

- Uses vectored exception handling
- Has a separate set of mode-specific registers for each exception mode
- Special instructions exist for storing and reloading processor state to and from mode-specific registers

## When an Exception Occurs

- Status register is copied into the mode-specific counterpart
- If processor is in Thumb state, change to ARM
- Store return address in mode-specific LR
- Store appropriate handler address in PC

## Returning from Exceptions

- Copy mode-specific status register back into user status register
- Copy return address from mode-specific LR back into PC

## System Control Coprocessor

- A set of 32 readable and writeable registers
- Which registers can be written depends on whether the processor is in the Secure Monitor operating mode
- Used in configuring various operations ranging from DMA to cache control to performance measurement

## SIMD Capabilities

- Vector Floating-Point coprocessor performs operations on 8 single-precision or 4 double-precision values simultaneously, in parallel with CPU
- Otherwise costly arithmetic operations such as square root are built into hardware
- Dedicated interface to main processor
- Results of compare instructions are stored in CPU status register
  - Could make interface bandwidth and latency a bottleneck

## VFP Pipelines

- Each pipeline shares decode and issue stages, but otherwise works independently and in parallel
- Multiply and ACcumulate (FMAC) pipeline has 7 execution stages
- Divide and Square root (DS) pipeline has 4 execution stages
- Load/Store (LS) pipeline has 1 execution and 2 memory access stages, and is responsible for communicating with the main processor

## VFP Registers and Data Formats

- Four banks of 8 32-bit registers
- Each instruction operates on a bank of registers
- Consecutive pairs of registers can store double-precision floating-point data
- Floating-point data format follows the IEEE standard